There's No Trick, Its Just a Simple Trick: A Web-Compat and Privacy Improving Approach to Third-party Web Storage

Jordan Jueckstock
North Carolina State University
jjuecks@ncsu.edu

Peter Snyder
Brave Software
pes@brave.com

Shaown Sarker
North Carolina State University
ssarker@ncsu.edu

Alexandros Kapravelos
North Carolina State University
akaprav@ncsu.edu

Benjamin Livshits
Brave Software
ben@brave.com

Abstract

While much current web privacy research focuses on browser fingerprinting, the boring fact is that the majority of current third-party web tracking is conducted using traditional, persistent-state identifiers. One possible explanation for the privacy community's focus on fingerprinting is that to date browsers have faced a lose-lose dilemma when dealing with third-party stateful identifiers: block state in third-party frames and break a significant number of webpages, or allow state in third-party frames and enable pervasive tracking. The alternative, middle-ground solutions that have been deployed all trade privacy for compatibility, rely on manually curated lists, or depend on the user to manage state and state-access themselves.

This work furthers privacy on the web by presenting a novel system for managing the lifetime of third-party storage, "page-length storage". We compare page-length storage to existing approaches for managing third-party state and find that page-length storage has the privacy protections of the most restrictive current option (i.e., blocking third-party storage) but web-compatibility properties mostly similar to the least restrictive option (i.e., allowing all third-party storage). This work further compares page-length storage to an alternative third-party storage partitioning scheme inspired by elements of Safari's tracking protections and finds that page-length storage provides superior privacy protections with comparable web-compatibility.

We provide a dataset of the privacy and compatibility behaviors observed when applying the compared third-party storage strategies on a crawl of the Tranco 1k and the quantitative metrics used to demonstrate that page-length storage matches or surpasses existing approaches. Finally, we provide an open-source implementation of our page-length storage approach, implemented as patches against Chromium.

1 Introduction

Web trackers use a variety of techniques to track and violate privacy on the web. Tracking is usually done through a mix of stateful tracking (i.e., storing and transmitting unique identifiers in the browser) and stateless tracking, or fingerprinting (i.e., attempting to uniquely identify a browser based on unique configuration and execution environment characteristics).

Though much recent privacy work has focused on stateless, fingerprinting-based tracking, we expect that the majority of tracking is still done using traditional stateful methods. This intuition is based on multiple factors, such as adtech uproar over Google's recent announcement [2] to stop sending cookies (only one of many ways of storing identifiers) to third-parties in the future, prior research demonstrating the popularity of storage-based tracking [22, 24, 41, 42, 49, 52], and expert insight from browser developers.

While the privacy community has had some success in designing defenses to stateless, fingerprinting tracking that protect users without breaking benign, user-serving page functionality [31, 37], researchers, industry and activists have been less successful in designing practical, robust defenses against webscale stateful third-party tracking.

Despite the press and attention that the "end of third-party cookies" has received, blocking third-party cookies (i.e., not sending cookies on requests for third-party sub-resources) does not provide any fundamental protections against stateful third-party tracking. Blocking third-party cookies is a positive step for web privacy, but because it prevents categories of accidental tracking or information disclosure, not because it prevents intentional tracking. Third-party frames can access the same cookies[1], `localStorage`[2], `indexDB`[3], or other JavaScript accessible storage methods (sometimes collectively called "DOM Storage"). In short, blocking third-party cookies is a necessary, but insufficient part of solving the general

---

[1]For completeness, we note that this isn't completely true, and that `HttpOnly` cookies cannot be accessed from JavaScript. But since `HttpOnly` doesn't provide protection against intentional tracking (since such trackers could just omit the `HttpOnly` instruction), we don't consider `HttpOnly` further in this work, and omit it from further discussion for concision.

[2]https://html.spec.whatwg.org/multipage/webstorage.html#the-localstorage-attribute

[3]https://www.w3.org/TR/IndexedDB-2/

problem of preventing stateful third-party tracking.

Though some browser vendors have taken some steps to address third-party stateful tracking, each approach has significant shortcomings and limitations. The details of each technique are described in Section 2.3, but at a high level, deployed approaches are incomplete and insufficient, either because they depend on curated lists and heuristics (i.e., Firefox and Edge), address tracking across sites but not time (i.e., Safari), defer the question to non-expert users (i.e., Storage Access API[4]), or provide strong protections against tracking but break sites for users (i.e., Brave).

We argue that practical, robust protections against stateful, third-party tracking should have at least three properties.

1. **Cross-site protection**: prevent third-parties from using stored identifiers to link browsing behavior across first-party sites.

2. **Cross-time protection**: prevent third-parties from using stored identifiers to link browsing behavior on the same first-party site across time.

3. **Web Compatibility**: not effect, or minimally impact, user-serving, non-privacy harming behavior in third-party frames.

In this work we aim to improve web privacy by presenting a new method of managing and limiting third-party state that we call "page-length storage". Section 3.1 presents the approach in detail, but at a high level, "page-length storage" is the unique combination of two features:

1. *page-length storage partitions third-party state by the top level document*. If a browser tab has loaded a page from origin A, and that page includes two sub-documents (i.e., `<iframe>`s) from origin B, the two sub-documents see the same storage, but different storage than B sees when B is the top-level document, and also different storage than origin B sub-documents on other pages and tabs.

2. *page-length storage sets the lifetime of all third-parties state to be equal to the lifetime of the top level document*. If a page from origin A opens and closes `<iframe>`s from origin B, all of those origin B frames see the same storage, even between frames being opened and closed. However, once the top-level page is closed, all the partitioned storage for B is cleared as well. Revisiting or reloading the top-level page will result in the B frames seeing empty storage.

Contributions. More concretely, this work makes the following contributions to improving privacy on the web.

---

1. The design of page-length storage, **a novel approach to managing third-party state** in web pages that provides strong privacy protections without breaking websites.

2. New, general **metrics for measuring the privacy and web-compatibility** properties of third-party storage policies.

3. An **open-source, prototype implementation** of page-length storage as a set of patches to Chromium [13].

4. A public **dataset of applying four storage policies to the Tranco 1k [43]**, a research-focused ranking of popular sites. Our dataset [13] includes the above privacy and compatibility metrics generated from four policies, each approximating a third-party storage policy currently deployed in a popular browser.

2 Background & Motivation

Modern browser technologies and the security policies that govern them are complex, so we must clearly define our terms and provide essential background on browser storage policy, user tracking techniques, and the state of the art in tracking countermeasures.

2.1 Same-Origin Policy & Storage Basics

Sites & Origins. Browsers isolate storage (e.g., cookies, `localStorage`, `indexDB`) according to the Same-Origin Policy (SOP) [11]. The SOP has grown complex, multifaceted, and inconsistent [46], and applies to many aspects of the web; here we describe only its most basic and universal elements, particularly as they relate to storage. An *origin* comprises a scheme (e.g., `https`), a complete DNS hostname, and an optional TCP port number. All state-impacting activities in a browser are associated with an origin derived from some relevant URL. For example, a script's execution origin is derived from the URL of the frame in which the script executes, and an HTTP request origin is derived from the URL being fetched.

Many activities are restricted to *same-origin* boundaries. For example, a script executing in origin A cannot access cookies stored for origin B. This is true even when a sub-document from origin B is embedded in a page from origin A. Storage is strictly isolated according to SOP: scripts can access cookies and DOM storage (e.g., `localStorage`) only for their execution origin, and HTTP requests store and transmit cookies only for their destination origin.

First and Third Parties. We now define two terms used through the rest of this paper, first-party and third-party. These terms are not unique to this work, but are frequently used to mean similar but not-quite-the-same things in research and web standards, so we define their use in this work explicitly.

---

[4]https://developer.mozilla.org/en-US/docs/Web/API/Storage_Access_API
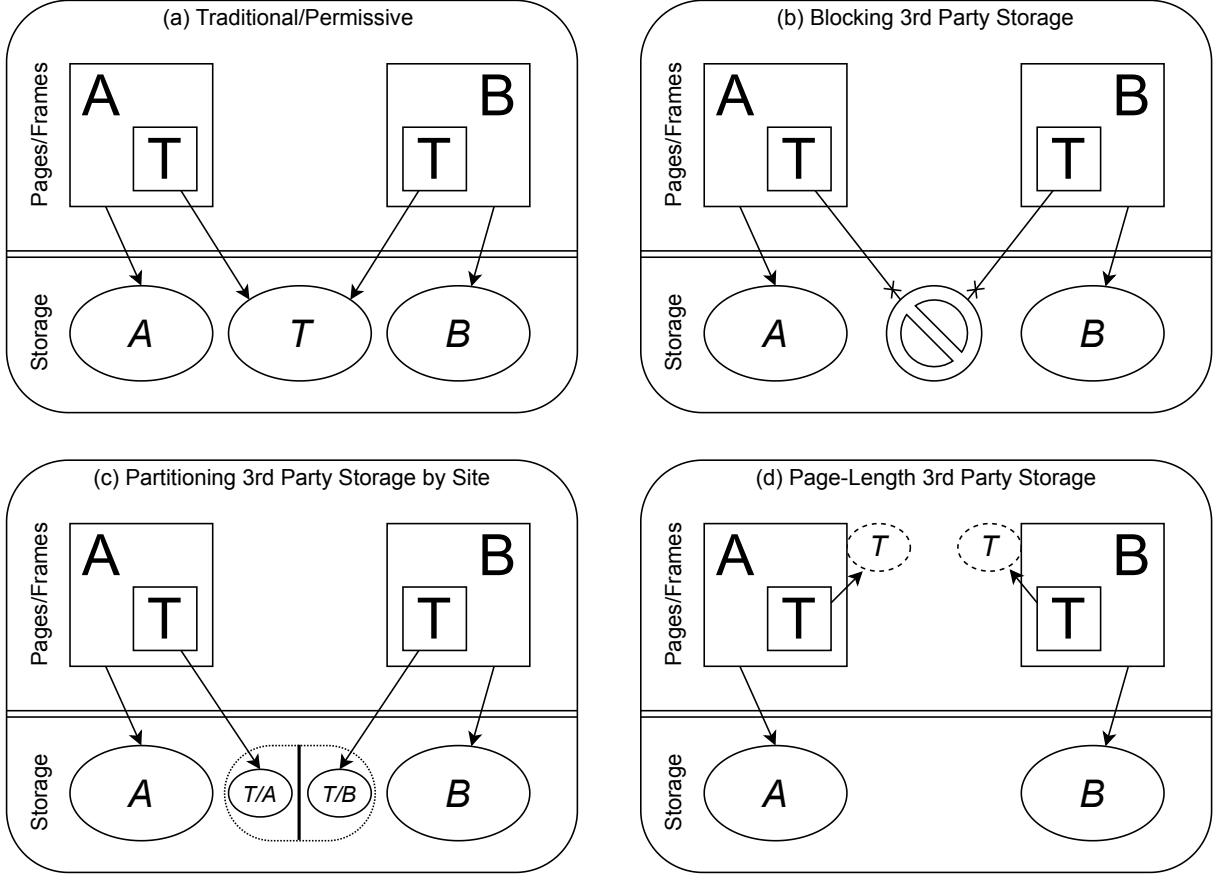
Figure 1: Third-party storage (a) fully allowed, (b) fully blocked, (c) partitioned by first-party context, and (d) scoped to hosting page life time (our proposal). *A, B, & T are distinct domains; T is embedded as a third-party within A & B.*

When loading a website, the **first-party** is the "site" portion of the top level document. This is the eTLD+1 of the URL shown in the navigation bar of the browser. Any sub-resources or sub-documents included in the page are considered first-party if they're fetched from the same eTLD+1 as the top level document. **Third-parties**, then, are any site not equal to the top-level document. A sub-document (i.e., <iframe>) is considered third-party if it is fetched from any origin not-equal to the top-level document, a script is third-party if it was fetched from a site different than the top level document, and so forth.

Finally, we note that when applying SOP to the web, what determines storage access is the "site" of the frame including a script, not the script itself. So if a page from origin A includes a script loaded from origin B, the script is third-party, but has access to the first-party storage. What storage area a script has access to is determined by the "site" of the page, not the "site" of the script.

2.2 User Tracking

Types of Behaviors Tracked. This work uses the term "tracking" to refer to a third-party re-identifying a visitor across visits to first-party sites. Unless otherwise specified, we use "tracking" to refer both to **cross-site tracking** (i.e., a third-party can link an individual's behavior across first-parties) and **cross-time tracking** (i.e., a third-party can identify the same person returning to the same first-party across sessions).

Stateful Tracking. The oldest, simplest and most common form of online tracking is "stateful" tracking, where a third-party stores a unique value on the user's browser, and reads that value back across different first-parties. While the terms *explicit* and *inferred* used by Roesner *et al.* [45] appear more precise, as both techniques involve state of some kind, the *stateful*/*stateless* terminology popularized by Mayer and Mitchell [35] appears dominant in subsequent research.

In the simplest case, stateful tracking works as follows. Sites A and B both include an iframe embedding site C. When the embedded site C is loaded, it looks to see if a unique identifier has been set, and if not it generates and stores one,

using any of the storage methods provided by the browser (cookies, `localStorage`, `indexDB`, etc). Embedded site C then checks to see what site's its being embedded in, and sends a message back to site C, recording that the same user visited both sites A and B.

Stateful tracking, at root, relies on a site being able to access persistent state in different contexts, and using the persistently stored state to link (conceptually) unrelated behavior. To build on the previous example, site C is able to track the user across A and B because C seems the same storage values when embedded in sites A and B, even though site A might not have any direct relationship with site B.

As we will discuss at length later, approaches for preventing stateful tracking involve either preventing third-parties from storing values, proving third-party different stored values when embedded in different contexts, or combinations of the two.

Other Tracking Techniques. While stateful tracking is the simplest, and likely the most common, method for tracking users online, there are other ways third-parties track users. While this work focuses on stateful tracking, in this subsection we briefly discuss these other, non-stateful techniques here for completeness:

- *Browser fingerprinting* refers to uniquely identifying a browser (or browser user) not through the storage and transmission of a unique identifier, but by identifying unique characteristics of the browser's configuration (e.g., plugins, preferred language, "dark mode") and execution environment (e.g., operating system, hardware capabilities).

- *Server-side tracking* is a broad term that loosely means tracking users across sites not through stored identifiers (i.e., stateful tracking) or unique configuration (i.e., fingerprinting), but through information the user provides to the site. For example if a user uses the same email address when registering on two different sites, a tracker could later use the repeated email address to link the users behavior across sites.

2.2.1 Focus on Stateful Tracking

This work presents a novel solution for preventing stateful cross-site and cross-time tracking. We aim to improve protections against stateful tracking because we think its where browsers are most lacking practical, robust, comptable defenses. While significant research has gone into building web-compatible defenses against stateless tracking (e.g., [31, 37]), the existing techniques for preventing stateful third-party tracking are either incomplete (i.e., they still allow significant privacy harm to occur) or incompatible (i.e., they break a significant number of websites).

2.3 Deployed Stateful Tracking Defenses

Real-world countermeasures currently deployed in production browsers illustrate a range of possible tradeoffs between privacy and compatibility. We rely heavily on the community-curated Cookie Status project [3] for up to date policy implementation details.

Brave: Block all Third-Party State. Brave [16], a Chromium fork featuring aggressive privacy protections called "Shields", defaults to blocking all forms of third-party storage. The officially correct way to block persistent third-party storage involves raising a JavaScript exception on script access to blocked storage APIs [10]. Few sites are prepared to handle these exceptions, however, and Brave improves compatibility by instead simply turning blocked third-party storage API accesses into no-ops. Brave also uses a whitelist to allow a small number of high-profile third-parties to use persistent storage in the context of specific first-party sites (e.g., *googleusercontent.com* when embedded from *google.com*) [7]. Brave's approach results in strong privacy protections at the cost of a higher incidence of site breakage, which may require users to selectively lower its Shields on incompatible sites.

Safari: Partition Third-Party State. Apple Safari [15] features Intelligent Tracking Prevention (ITP), a combination of storage restriction policies, opt-in APIs, and on-client classification of tracking domains via machine-learning [12]. Safari never transmits cookies on third-party HTTP requests. Cookie and `localStorage` access in third-party frames are partitioned on first-party site identity to prevent stateful lateral tracking (as in Figure 1c). Safari provides developers with a `requestStorageAccess` API to request user permission to access unpartitioned third-party storage across first-party contexts. This opt-in approach allows users to accept the potential for lateral tracking in exchange for useful functionality such as cross-site login state.

Safari features a number of additional policies and heuristics to restrict the lifetime of items stored by domains ITP has classified as probable trackers. These restrictions impact but do not categorically eliminate potential for longitudinal tracking by third-parties. The Safari ITP approach provides strong cross-site tracking protections while avoiding full-blocking with its associated site breakage, but it does not eliminate across-time tracking by third-parties, and the non-deterministic impact of machine-learning on its policy enforcement can make it challenging for web developers to reason about.

Firefox and Edge: Restrict Known Bad Actors. Mozilla Firefox [6] has adopted a selective storage policy that depends on the Disconnect [8] list of curated tracking domains. In general, third-party origins not found in the Disconnect list are granted unrestricted access third-party storage. Third-party origins classified as trackers by Disconnect are given access to third-party storage on the first five first-party sites embedding

that third-party origin. Subsequent additional sites embedding that third-party origin will result in user opt-in prompts to allow third-party storage which must be accepted to allow use of third-party storage by that origin on that first-party site.

Exceptions to these restrictions are made for first-party domains identified by the Disconnect list as related to specific third-party origins (e.g., *googleusercontent.com* embedded on *google.com*). The Firefox approach is one of compromise: well-known tracking domains face restrictions on the reach of their lateral tracking, but protection depends heavily on the validity and coverage of the underlying filter list.

Microsoft Edge [4] has begun to deploy filter list-based storage restrictions similar to those performed by Firefox, with all the benefits and drawbacks of this compromise approach summarized above.

Chrome: Unrestricted Third-Party State. Google Chrome [9], in contrast to all of the above, permits full third-party storage use, including sending cookies on HTTP requests to third-party resources. Google has announced intentions to phase out third-party cookie support [2] in the near future; technical details remain vague, but their wording implies eliminating only cookies on third-party HTTP requests, not restricting third-party storage in general. Chrome dominates as the world's most popular browser for both desktop and mobile markets [1], understandably prompting web developers to target its behavior for maximum compatibility, and indirectly perpetuating the status quo of stateful tracking in the process.

2.4 Compatibility and Tracking Protections

Finally, we present some ways that existing protections against third-party stateful tracking break websites. We present these as moderating examples, and useful constraints, in designing page-length storage. Without considering these compatibility concerns, solutions will tend to simplistic, "block-everything" approaches that end up not being useful, and so not being effective in protecting privacy.

We gather the following examples from Brave's public issue tracker[5]. We pull from Brave's breakage reports since Brave has the most aggressive restrictions on third-party storage of the surveyed browsers, and so the largest number of storage-related compatibility problems. Nevertheless, we present these as examples of the kinds of compatibility problems that third-party storage protections can cause.

2.4.1 Uncaught Exceptions from Blocking Storage

Strict third-party storage blocking breaks embedded SlideShare slide show widgets (e.g., `https://support.blogactiv.eu/2015/04/24/how-to-embed-slideshare/`) on Chrome. The widget becomes inert, not
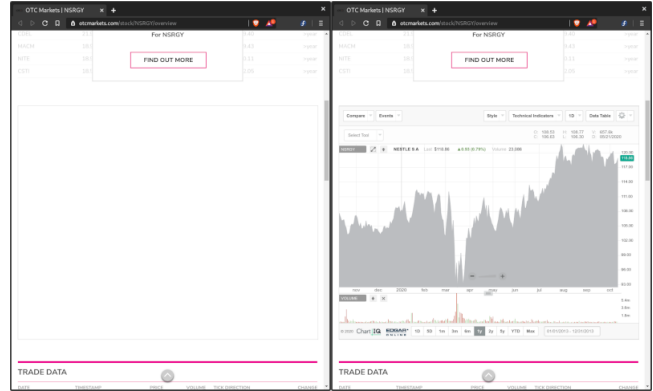

Figure 2: Stock market graph broken by strict third-party storage blocking (left) and working with page-length storage (right).

responding to clicks, when Chrome's implementation of strict third-party storage blocking (correctly, per the specification) raises JavaScript exceptions on access to storage APIs. Brave's silent no-op implementation of strict third-party storage blocking is sufficient to prevent breakage in this case; successful storage access is clearly not essential to this widget's functionality.

A similar example is provided by a data plot widget broken by strict third-party storage blocking (e.g., `https://www.otcmarkets.com/stock/NSRGY/overview`). Once again, strict third-party storage blocking causes a JavaScript run-time error which results in a blank data plot (see Figure 2). In this case, Brave's silent no-op blocking does not help: the error is caused by property access on a null value returned from a no-op API stub.

2.4.2 Breaking Cookie-Based Third-Party Sessions

A server-side example of strict third-party storage blocking causing breakage is provided by a live code editing/running widget embedded in the R language documentation (e.g., `https://www.rdocumentation.org/packages/grid/versions/3.6.2/topics/grid.plot.and.legend`). The embedded widget tries to establish a cookie-based session with third-party domain *multiplexer-prod.datacamp.com*. Failure to persist third-party cookies results in HTTP 403 errors on subsequent HTTP requests, preventing code execution and output display.

A broken video player on a popular commentary and analysis site provides another example (`https://fivethirtyeight.com/videos/do-you-buy-that-biden-should-pick-a-running-mate-from-a-swing-state/`). With third-party storage blocked, the video player remains blank indefinitely. In this case, the video player functionality is broken because the frame attempts to use `localStorage` to persist values across pages.

3 Design & Implementation

We propose and prototype a novel browser storage policy that prevents both cross-site and cross-time stateful tracking while measurably improving site compatibility over traditional third-party storage blocking. Page-length storage prevents stateful third-party tracking while minimizing site breakage by making third-party storage fully functional within a strictly isolated, ephemeral scope. We also provide a brief overview of how we developed and tested our page-length storage prototype within a Chromium-based browser.

3.1 Policy Design

The key insight behind page-length storage is that site breakage can be minimized without compromising privacy by making all interactions with third-party storage behave normally (i.e., permissively), but only within the isolated, ephemeral scope of the containing page's lifespan. The containing page is the top-level frame, loaded from the URL displayed in the browser's navigation UI (e.g., address bar). Its lifespan expands from the moment the top-level frame committed to loading that document URL to the moment any navigation event (including even reloads of the same URL) discards the contents of the top-level frame.

Within the isolated, ephemeral scope of each top-level page object's lifespan, third-party storage access behaves normally for both scripts executing in third-party <iframe>s and HTTP requests to third-party domains. SOP enforcement is unchanged. All storage behaves in the traditional, permissive way with one exception: third-party storage starts out empty and is discarded along with the top-level page object on top-frame navigation. Any site embedding third-party content which functions correctly under permissive policy for first-time visitors with empty cookie jars should function correctly under page-length storage policy.

Isolating third-party storage to single page lifespans provides a good tracking *vs.* compatibility compromise. Page-length storage prevents stateful cross-site and cross-time tracking automatically, as third-parties cannot "remember" anything past top-level page (re-)loads. Compare Figures 1a and 1d. As a practical matter, third-party content cannot silently manipulate top-level page navigation, so it cannot test whether third-party storage will persist beyond top-level navigations. All tests that can be done silently within the scope of a single page's lifespan will appear fully functional, as for a first-time visitor with uninitialized third-party storage.

Some hypothetical examples illustrate the impact of this approach.

First, consider two <iframe>s from the same third-party embedded on a single page document: these will share the same ephemeral third-party storage partition and so can use all forms of third-party storage, via both script access and HTTP cookies, to communicate with each other and the remote third-party origin for the duration of the embedding page's lifespan.

Second, consider a third-party <iframe> embedded on a page that is loaded on two different tabs simultaneously: each instance of the frame is using its parent-page's ephemeral third-party storage partition, so no cross-site stateful sharing/communicating is possible between them.

Third, consider a third-party <iframe> embedded on a page that is loaded and then reloaded in the same tab: each page load (regardless of URL) discards the previous page's ephemeral third-party storage partition, so no cross-time stateful sharing/communicating is possible.

Finally, consider a third-party <iframe> embedded in two pages hosted on different first-party domains: whether these pages are loaded sequentially in one tab, or simultaneously in two tabs, each third-party frame is using its own parent-page's ephemeral third-party storage partition, so again no cross-site stateful sharing/communicating is possible.

3.2 Prototype Implementation

We implement our page-length storage prototype as a set of patches to Brave 1.12.48 (based on Chromium 83). We use Brave, and version 1.12.48 specifically, in order to use the latest revision of PageGraph for data collection, per Section 4.1.4. However, our patches are completely independent of PageGraph's patches and can be built without them present. The most relevant change from stock Chromium provided by Brave is its gentler approach to third-party storage blocking, which it enables by default. Instead of raising a JavaScript exception on script access to blocked storage (per the specification), Brave makes the access a silent no-op, returning a null value.

Chromium's standard architecture includes content and storage isolation mechanisms relevant to our design goals. In addition to classic SOP enforcement, Chrome isolates content rendering and JavaScript execution into separate render processes partitioned on a same-site basis (see Section 2.1). Each render process uses **one** storage partition, which can be persistent (the default) or ephemeral (private mode), and which can additionally be partitioned by arbitrary identifiers (for Chrome apps and extensions). HTTP traffic is managed by a dedicating networking process, which chooses a storage partition for HTTP cookies based on the frame initiating the request.

We exploit this existing site storage isolation framework to prototype page-length storage with minimal changes to the browser. Our classification of frames and requests as third-party reuses the same-site logic already in Chromium and is always relative to the top-level page URL (not <iframe> URLs). Each time a tab's top-level frame loads a page URL, we generate and store a UUID identifying that load event (the *load key*). When third-party frames are subsequently created and assigned to separate render processes, we augment the third-party site identifier with the top-level frame's current

load key to enforce page-level isolation between ephemeral third-party storage partitions. HTTP requests to third-parties are bound to the associated ephemeral third-party storage partition, which is created on demand if necessary. The result is unchanged first-party storage behavior, and fully functional third-party storage that lives only as long as the containing page document, as in Figure 1d.

3.3 Implementation Remarks

The changes required to prototype page-length storage proved deceptively small. A total of 276 lines of C++ were added, changed, or removed by our patches. The scale of these patches, small relative to the millions of source code lines of Chromium, belie the challenge of finding the right places to patch. Most changes relating to storage partition creation and isolation were confined to the main "browser" process in Chromium's multi-process architecture, which has access to the entire frame tree for each tab, and were thus relatively straightforward to implement. Binding third-party HTTP requests to isolated, ephemeral storage partitions on demand, however, crossed process boundaries into the network process, which does not have access to frame tree context information, and required additional IPC messaging and timing concerns.

The implementation demonstrated correctness and robustness fully sufficient for prototype testing. The available Chromium unit tests all passed, except for a few implementation-specific assertions we expected to fail after our changes. Manual testing of multiple scenarios like the examples from Section 3.1, included nested third-party `<iframe>`s, showed expected behavior in all cases. Furthermore, the prototype's error rate during automated crawls were favorably comparable to stock permissive policy (see Section 5.1).

Prototype performance proved adequate despite not being a design priority. Because performance was not a design priority for the prototype, we did not perform any benchmarks. In theory, our approach should reduce performance over stock Chromium: it can produce more render processes, can involve more I/O operations creating temporary directories, and definitely invokes additional IPC overhead between network and render processes. However, both manual testing and automated crawling with the prototype revealed no obvious performance degradation. Furthermore, none of these issues are inherent in the policy itself, and there is no reason to believe a performance-tuned production implementation would produce any significant overhead compared to traditional policies.

4 Methodology

We evaluate our proposed policy by comparing its tracking and compatibility performance against alternative policies during automated, stateful crawls of popular web sites.

4.1 Crawl Methodology

Here we describe our data collection procedure in sufficient detail to permit straightforward experiment reproduction.

4.1.1 Target URLs

We generated a seed list of URLs to visit in parallel using a stateless *pilot crawl* of the Tranco 1k sites [43]. To achieve depth and representative sampling of web content, we must explore more than just the "landing page" of each site. But each of our 8 parallel crawls must visit the same sequence of page URLs to produce comparable results. Coordinating the link spidering and selection process across parallel crawls introduces needless engineering complexity. Our solution was to perform a stateless pilot crawl using stock Brave to visit the Tranco 1k sites' landing pages and spider three links deep into the site structure. This approach, using Tranco list snapshot JZZY, produced 3,419 total deduplicated page URLs to visit.

4.1.2 Policy Variants

We collect data using four distinct policy variants:

- **Permissive:** Allows all forms of third-party storage, as per Figure 1a. Stock Chrome behavior. Presumed to cause no breakage.

- **Strict third-party storage blocking:** Blocks all forms of third-party storage, as per Figure 1b. Treats access as no-op. Presumed to cause the most breakage.

- **Site-keyed:** Partitions persistent third-party storage by first-party eTLD+1, as per Figure 1c. Alternative to our proposed policy, inspired by elements of Safari ITP.

- **Page-length:** Isolates third-party storage in ephemeral, per-page partitions, as per Figure 1d. Our proposed policy.

4.1.3 Crawl Execution

We executed our stateful crawls in parallel across all storage policies without any simulated user interactions. We deployed two instances of each tested policy to verify behavioral consistency and provide similarity-score baselines (see Section 4.2.3). The crawlers maintained independent, persistent user profiles for each policy instance to maintain realistic state across all sequential page visits. The full experiment included 2 iterations crawling the master URL list to provide data on cross-time tracking across repeat visits. All crawls were performed in parallel and simultaneously (but without active synchronization between profiles) from a single network vantage point. Each page visit was performed in a freshly launched, non-headless (i.e., rendering to the *Xvfb* headless display server) browser instance. Navigation was

allowed to time out after 30 seconds. Assuming no navigation timeout, our crawlers waited for 30 seconds after the `DOMcontentloaded` event (i.e., main document fetched and parsed but subresources not fully loaded yet) before tearing down the browser instance. No simulated user interactions were attempted.

4.1.4 PageGraph Instrumentation

We use PageGraph, an instrumentation system built into an experimental branch of Brave, to record internal page behaviors. PageGraph patches the V8 JS engine and the Blink HTML rendering engine to capture and annotate a graph of each HTML document's DOM structure and the events that constructed and modified it. Nodes represent entities such as DOM elements, scripts, HTTP resources, storage mechanisms, and a selective subset of builtin and DOM-provided JavaScript APIs. Edges represent relationships between nodes such as DOM structures and script interactions with DOM elements, DOM events, JavaScript APIs, and HTTP requests. The set of non-structural edges in each of these graphs constitute the dynamic behaviors of the originating page. Behavioral-edge-set similarity can be quantified using Jaccard index scores to provide a useful proxy for behavioral compatibility among compared storage policies.

4.2 Evaluation Methodology

We evaluate our proposed policy's privacy and compatibility performance using full-scale quantitative stateful tracking metrics, full-scale quantitative site behavior similarity metrics, and randomly-sampled qualitative assessment of site breakage.

4.2.1 Preliminary Data Filtering

We focus our analysis on frames of interest: i.e., third-party frames not flagged as advertisements. Our classification of third-party *vs.* first-party frames is based on eTLD+1 matches derived from the Public Suffix List [14]. Frames loaded from the same eTLD+1 as the main page URL are first-party frames; all others are third-party frames. We eliminate from consideration all first-party frames and third-party frames flagged as advertising content by the community-maintained EasyList [5]. This filtering eliminates noise from our evaluation: first-party storage is not affected by our policy change, and we are unconcerned about breakage of known advertising content.

4.2.2 Quantitative Privacy Assessments

Tracking Potential. The central metric we use to quantify potential for stateful cross-site and cross-time tracking by third-parties is the *potentially identifying cookie flow* (PICF).

A *cookie flow* is the combination of an HTTP cookie and a third-party eTLD+1 receiving that cookie. We consider cookie flows *potentially identifying* when the cookie values meet a tunable minimum size threshold and are **unique** to a single browser profile during our stateful crawls. There are other forms of third-party storage available, and other channels by which identifying tokens can be transmitted to third-parties. But we use cookies as a representative measure of stateful tracking because they are unambiguous in structure, ubiquitous as tracking IDs, and essentially unrestricted by stock Chrome, our baseline. (Both our page-length storage and site-keyed implementations apply their storage policies to **all** forms of third-party storage, not just cookies.)

Cross-Site Tracking. Identical PICFs seen across multiple distinct top-level sites visited represent potential for cross-site tracking by the associated third-party domain. We aggregate cross-site PICFs to count the total number of top-level sites across which each distinct third-party domain seen could have tracked our crawler profiles, giving us summary scores of "cross-site trackability" by which to compare all our storage policies. These scores can be visualized using cumulative sum curves, as shown in Section 5.2.

Cross-Time Tracking. PICFs seen on a given top-level site across multiple pages/crawls represent potential for cross-time, or visit-to-visit, tracking by a given third-party domain. We aggregate cross-time PICFs to count the total number of third-party domains which could have tracked our crawler profiles for each distinct top-level site domain visited, giving us summary scores of "cross-time trackability" by which to compare all our storage policies. These scores can be visualized using cumulative sum curves, as shown in Section 5.3.

4.2.3 Quantitative Compatibility Assessment

We assess site compatibility across storage policies using a quantifiable proxy measure: similarity of internal page behaviors as reported by PageGraph. Our insight is to presume no storage-based breakage for permissive profiles and some unknown (but non-zero) amount of breakage on strict third-party storage blocking profiles. If alternative policy (e.g., page-length storage) profiles produce content behaviors more similar to the permissive baseline than do the strict third-party storage blocking profiles, then the alternate policy is less likely than strict third-party storage blocking to cause breakage.

We model and compare content behaviors using the set of non-structural (i.e., action or event) edges in PageGraph representations of relevant frames. Similarity between edge sets can be measured using the Jaccard index: $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$. Index scores range from 0 (no intersection) to 1 (equality). We consider the score undefined when both sets were empty.

We compare content behaviors across identical frames loaded on identical pages across all tested policies. Frames

and pages are identified and matched by full URL. The similarity score of the two permissive profiles provides the compatibility baseline: the presumed best-possible similarity score for that frame/page instance. The other profiles are each compared with a single permissive profile to provide similarity scores to compare against the baseline. The cumulative sum of all frame/page instance similarity scores for each profile can be visualized to show which policies track closest to the baseline across all visited pages (see Section 5.4).

We optimized the set of PageGraph node types included in our behavioral sets to maximize the distance between strict third-party storage blocking policy scores and the permissive baseline score. Our intuition is that the baseline score provides a threshold of "reasonable" behavioral differences between two different instances of the same content loaded in different browsers at about the same time. The farther away from this baseline a policy scores, the greater the likelihood of unreasonable, or breaking, differences in behavior.

We identified 11 PageGraph node types relevant to behavioral analysis, a set small enough to be amenable to brute force optimization across its power set. Optimization relied on a random sample of 100 frame/site instances extracted from a preliminary full-scale crawl dataset, whose unoptimized similarity curves matched those of the entire data set, indicating a representative sampling. On this data subset we tested the strict third-party storage blocking separation from the permissive baseline for every subset of relevant PageGraph node types. The results confirmed our intuition that the least helpful node types were structural elements like HTML elements and DOM text blocks; less intuitively, they also showed that Page-Graph's set of instrumented DOM manipulation JavaScript APIs was similarly unhelpful. The final optimial node type set comprised scripts and PageGraph's selected JavaScript builtin APIs (e.g., date functions), HTTP resources, frame structures (DOM roots and frame-owning elements), and storage mechanisms (cookie jars, local and session storage buckets). Only edges (i.e., behaviors) linking these node types are included in the behavior similarity results presented in Section 5.4.

4.2.4 Qualitative Compatibility Assessment

We augment our quantitative proxy assessment of site compatibility with blinded multi-grader manual analysis for breakage within a random sample of sites loading popular third-party content. Our methodology is heavily inspired by a similar experiment by Snyder *et al.* [48].

To select our set of URLs to test, we first identified the most popular third-party, non-ad-blocked frame URLs within our crawl dataset. We sorted these by the harmonic mean of the number of pages embedding that frame and the number of third-party cookies set for the frame's eTLD+1. This metric is higher for frames which appear on a large number of sites and have access to a large number of cookies: prime candidates for testing third-party storage policy changes. We

selected the top 10 frame URLs with distinct eTLD+1s, to have higher content diversity. We further filtered out frames which appeared only on non-English sites (e.g., frames from *baidu.com* and *alicdn.com*), and frames which did not have a content type of either HTML or JavaScript (e.g., frames from *sharethis.com* with a content type of image).

For each of the 10 selected frame URLs, we randomly selected 5 candidate page URLs observed to embed that frame URL during our crawls, giving us 50 candidate URLs for manual analysis. Upon closer inspection of the frame contents, some frames did not have any real estate on the page and simply contained JS script, which would interact or render with DOM elements elsewhere on the containing site. With this insight, we adopted a holistic approach to evaluate breakage rather than simply observing the behavior of one frame.

We had two graders evaluate each of our candidate URLs for the policy variants in Section 4.1.2. The graders would visit a candidate URL first with a permissive profile, the Chrome default. This visit is our *control* visit for manual analysis. It was followed by a visit to the same URL with each of the site-keyed, page-length, and strict third-party storage blocking profiles. Every visit, including the control visit, was independent of all others, with a fresh browser profile to ensure no browsing state carried over between tests/visits. To keep our graders unbiased, subsequent visits to the candidate URL after the control visit were randomly coded so the graders did not know which profile they were using.

In our holistic approach to evaluation, each grader would visit the candidate URL with the control profile first. We instructed each grader to perform as many interactive actions on the candidate site within one minute, which is the average dwelling time for a typical web-user on a webpage [34]. Activities depended on category of site: on news portals, our graders would skim through, search for articles, watch embedded videos, click on ads, or try to sign-up for newsletters; on shopping sites, they would search for products, add products to the shopping cart, and initiate a checkout; on product sites, graders would either skim informational material, or try any video streams available, etc. Subsequently, the graders would visit the same URL with the 3 coded profiles, performing similar actions as during the control visit, observing any deviations from the control visit, and scoring their visit on a 1 to 3 scoring scale. The graders gave a coded profile visit a score of **1** if the visit did not have any perceptible deviations from the control; **2** if there were some deviations from the control visit, but this did not hinder their visiting experience or the tasks the graders attempted on the site; and **3** if the visit had significant deviations from the control, preventing the graders from replicating their control visit activities.

Given the highly subjective nature of the evaluation scheme, we carefully assess grader agreement. Our graders evaluated the candidate URLs independently, unaware of the other grader's scores. In our evaluation, our graders had a high agreement percentage (95.33%). We also com-
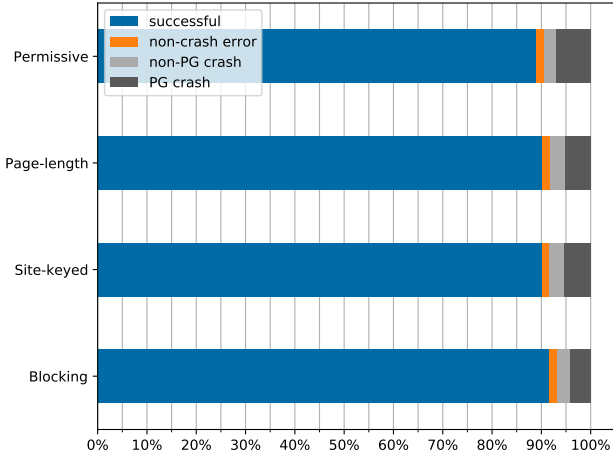
Figure 3: Crawl success rate varied modestly across policies but was always reasonably high.

Figure 4: Of our tested policies, all but permissive essentially eliminated stateful cross-site tracking potential.

puted the Cohen's Kappa inter-rater reliability statistic [27] as 0.69, showing statistically substantial agreement between our graders [36]. We present the results of our manual evaluation in Section 5.5.

5 Results

Our experimental results show that page-length storage combines best-case stateful tracking protection with near-best-case site compatibility.

5.1 Crawl Statistics

Our stateful web crawls ran from September 12-16 on a single Linux virtual machine (40 VCPUs, 100GiB RAM).Combined, the crawls visited 27,352 total pages using 8 user profiles and produced 280,219 PageGraph files (405 GB).

Error rates were acceptable (Figure 3) if somewhat amplified by PageGraph internal consistency assertion failures. PageGraph's instrumentation is expansive and tracks complex interactions between JavaScript execution, DOM manipulation, and network traffic. Whenever unexpected corner cases (or bugs) prevent it from establishing unambiguous context for an event or activity, PageGraph logs the issue and terminates the browser rather than recording unreliable data.

5.2 Privacy: Cross-Site Tracking Potential

Page-length storage eliminates stateful cross-site tracking as effectively as does strict third-party storage blocking. See Figure 4. The cumulative sum curves show the aggregate counts of sites across which third-parties could track users under different policies, calculated using the tracking-potential heuristics described in Section 4.2.2. Page-length, site-keyed,
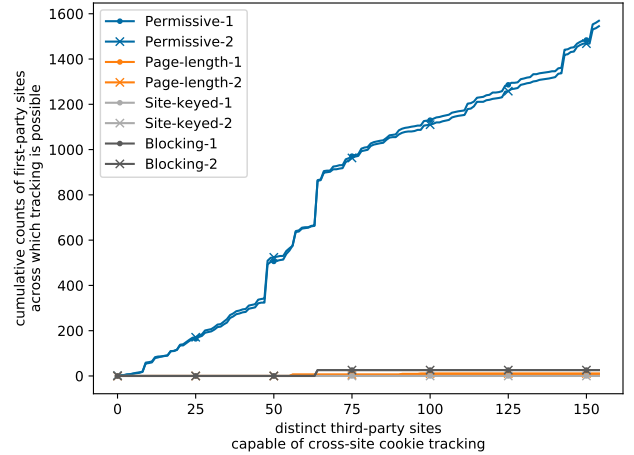
and strict third-party storage blocking policies are roughly equal at preventing stateful cross-site tracking. This result is logical and unsurprising: if third-party storage is not available (or is partitioned by first-party site, or is strictly ephemeral), it cannot be used to pass identifying state across site boundaries.

5.3 Privacy: Cross-Time Tracking Potential

Page-length storage also eliminates stateful cross-time tracking as effectively as does full third-party storage blocking, which is a significant improvement over site-keyed storage. See Figure 5. These curves show the cumulative sums of third-parties which could longitudinally track return visitors across the Tranco 1k sites, as described in Section 4.2.2. Unsurprisingly, permissive policy allows the most cross-time tracking; its strong cross-site tracking ability implies cross-time tracking ability. Persistent third-party storage, even if partitioned by first-party site context, is still accessible on repeat visits, allowing cross-time tracking. Thus, page-length and strict third-party storage blocking policies equally provide stronger cross-time tracking protection than site-keyed policy can.

5.4 Compatibility: Quantitative Assessment

Page-length storage produces page behaviors much closer to the permissive policy baseline than does full third-party storage blocking, as shown in Figure 6. These curves show cumulative sums of similarity scores between one of our permissive crawl profiles and all other profiles, normalized to show 1.0 as the maximum possible score (perfect similarity on all instances). The curve showing the similarity scores between the two permissive profiles provides a baseline (i.e., the best scores observed). Note the high consistency between all pairs of same-policy profiles. While even the baseline falls
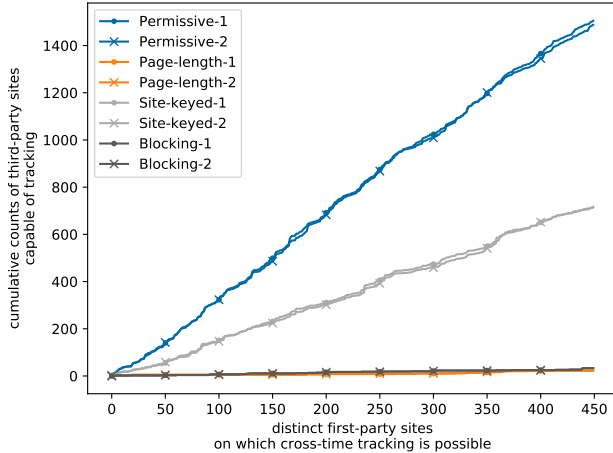
Figure 5: Our page-length policy significantly outperforms both permissive and site-keyed policies at reducing cross-time tracking potential.
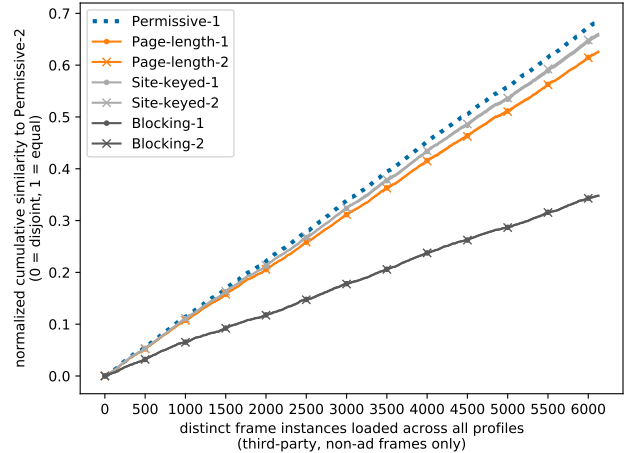

Figure 6: Our page-length policy produces page behaviors within third-party frames much closer to the permissive baseline than does the breakage-prone strict third-party storage blocking policy.

Profile	Pages Broken	% Broken (n=50)
Site-keyed	4	8%
Page-length	2	4%
Third-party blocking	5	10%

Table 1: Candidate URL breakage as assesses by holistic (whole-page) manual grading

short of perfect similarity, there is a clear signal in the grouping of policies. The strict third-party storage blocking policies produced the curves farthest from the baseline, as expected, well isolated from all the other policies. The non-blocking policies (site-keyed and page-length) both produced curves much closer to the baseline than to strict third-party storage blocking. The stark separation of curves strongly suggests that the non-blocking policies induce significantly less overall deviation from "normal" behavior (and thus less breakage) than does strict third-party storage blocking.

5.5 Compatibility: Qualitative Assessment

As described in Section 4.2.4, we had two graders independently perform manual evaluation on our set of 50 candidate URLs for each of the three profiles: site-keyed, page-length, and strict third-party storage blocking to manually assess each policy's potential for breaking sites. The graders independently evaluated each candidate site for each of the three profiles to find any deviations from our control profile, permissive, the Chrome default. The graders gave each visit a score on a scale of 1 to 3, as detailed in Section 4.2.4. We conservatively considered deviation from the control visit as a form of breakage, resulting in a score of greater than 1. We summarize the instances of graded breakage for each profile in Table 1. Grader notes on several reported breakages for page-length and site-keyed suggest that at least some of those deviations involved render process crashes rather than actual content breakage, possibly due to obscure bugs in those prototypes.

Considering the 5 breakages observed for the strict third-party storage blocking profile, the page-length profile either scored similar (2 cases) or improved (3 cases) in terms of raw

grader scores. In contrast to the site-keyed profile (4 breakages), the page-length profile again had either scored equal (2 cases) or better (2 cases). There were **no cases** where there was a breakage for the page-length profile with worse score compared to either of the strict third-party storage blocking or the site-keyed profiles. We concluded that the page-length profile performed reliably better than the strict third-party storage blocking profile, and that it performed as well or better than the site-keyed profiles. The observed rate of breakage for strict third-party storage blocking (10%) appears reasonable, and the roughly 2-to-1 advantage of page-length storage over strict third-party storage blocking observed in manual testing parallels a similar advantage in mean cumulative similarity score observed in quantitative analysis (Section 5.4).

6 Discussion

6.1 Limitations

The principal design limitation of page-length storage is the fact that some useful third-party webcomponents may simply require persistent, non-partitioned storage. We suspect that persistent storage for embedded third-party content is

more a matter of user convenience than essential functionality (e.g., customizing an embedded video player when the user is logged into the third-party site hosting the video). In any case, page-length storage can and should be augmented in production with the `requestStorageAccess` API to allow the user to opt-in to persistent storage for specific third-parties, either universally or on a specific first-party.

Our quantitative assessments of tracking and compatibility are subject to the limitations and risks of automated web crawls. While the scale of our crawl is modest, we believe the Tranco 1k provides a realistic sample of popular, mainstream web content and thus meets our evaluation needs. Spidering 3 links deep past landing pages likewise provides reasonable sampling of site content without exhausting our time and space budget, as PageGraph can generate large volumes of data per page. All our crawlers were stateful and non-headless, giving them a fair chance at evading the most trivial forms of bot detection. More sophisticated bot detection depending on "human" interactions with page content should treat all profiles identically (as bots; we performed no interaction simulations). We thus believe that whatever impact bot detection had on our crawlers, it would have affected all our profiles similarly and not significantly skewed our results.

6.2   Next Steps

Page-length storage can be further, better evaluated by real users by deploying it first to browsers serving privacy-conscious audiences. Production implementations will be somewhat more complex than our prototype (to address performance and maintainability concerns) but should require only modest investment by browser vendors. Production implementations should use the `requestStorageAccess` API to allow user opt-in to useful third-party storage access. Vendors could then deploy page-length storage to privacy-conscious users already blocking third-party storage and observe the impact on their site breakage reports.

Ultimately, page-length storage can be standardized to provide a near-best-of-both-worlds solution to the problem of persistent third-party storage abuse. Legacy content that assumes third-party storage access can be largely accommodated to minimize site breakage, without privacy loss. Modernized content can use the `requestStorageAccess` API to bypass page-length storage with user consent and gain controlled access to persistent third-party storage. The user wins: content that really needs third-party storage access to provide tangible benefit to the user can do so with the user's explicit permission, but the risk of permission denial and user alienation will motivate publishers of content providing less compelling user benefit (e.g., advertisers and trackers) to make do with less intrusive technologies.

7   Related Work

Stateful User Tracking. Storage-based user tracking, usually called "stateful" tracking and traditionally involving cookies, has been extensively studied. Mayer and Mitchell's seminal third-party web tracking study covered both stateful and stateless techniques and introduced the influential FourthParty web measurement framework [35]. A contemporary stateful tracking measurement work by Roesner *et al.* defined alternate terms "explicit" and "inferred" for stateful and stateless techniques, respectively, while measuring stateful tracking exclusively [45]. Acar *et al.*'s classic, large-scale user tracking measurement study emphasized stateful tracking and hinted at the nascent problem of cookie syncing [17]. A large-scale evaluation of stateful third-party tracking by Li *et al.* focused on cookies, the most prevalent form observed, and used machine-learning to identify third-party cookie tracking on 46% of the Alexa 10k [33]. Engelhardt and Narayanan's extremely large-scale user tracking measurement study included both stateful and stateless techniques, covered the entire Alexa Top Million sites, and introduced the widely used OpenWPM web measurement framework [22]. Yang and Yue recently extended classic tracking measurement methodologies to mobile web clients and reported distinctive but analogous groups of tracking domains compared to traditional desktop web tracking [50]. Despite the increasing sophistication of web tracking and countermeasure technologies, Fouad *et al.*'s recent exploration of obscure pixel-trackers showed classic third-party cookie tracking to still be effective and prevalent in the wild [24]. Zimmeck *et al.* even found traditional stateful tracking techniques to provide usable building blocks for cross-device tracking via linking together independent tracking sessions from different devices [52], a phenomenon conceptually similar to cookie syncing.

Cookie Syncing & Other State Transfers. Third-parties can collude to share stored user tracking identifiers and expand their tracking scope via *cookie syncing*. Olejnik *et al.* performed the first major measurement of cookie syncing in the wild, reporting that up to 27% of a user's browsing history could be leaked via cookie syncing [39]. Falahrastegar et al. measured distinctive personal identifiers and entities sharing them across the web, focusing on the groups engaged in sharing and how user behavior affects sharing [23]. Our procedure for selecting potentially identifying cookie flows shares some similarities with their selection of personal identifiers. Papadopoulos *et al.* identified cookie syncing as a major source of hidden costs to users imposed by digital advertising online [42]. Subsequent work documented the state of the art in cookie syncing, reinforcing the importance of third-party cookies to contemporary tracking [41].

Tracking identifiers can be passed across first-party domains using means other than stored state. Stopczynski *et al.* provide evidence that modern defenses like Safari ITP are effective but are being actively attacked and evaded, e.g.,

via abuse of HTTP redirects passing identifiers in modified URLs [49]. For the moment, these attacks appear to constitute efforts to reestablish traditional cookie tracking disrupted by ITP rather than the emergence of a new tracking paradigm.

Browser Fingerprinting. A major category of web privacy research for the past decade has involved stateless tracking via fingerprinting. The Panoptoclick project's seminal report on browser fingerprintability [21] popularized the threat as a potential tracking vector and launched a flurry of related research. Acar *et al.* measured fingerprinting in the wild and found it much more prevalent than commonly estimated at the time [18]. Olejnik *et al.* dissected the infamous and quickly deprecated Battery Status API as a particularly egregious source of fingerprinting entropy [38]. Laperdrix *et al.* identified new fingerprinting vectors from emerging desktop and mobile web technologies, but also identified potential trends toward reduced fingerprinting threats [32]. The current threat status of fingerprinting remains ambiguous: Gomez *et al.* reported findings that Panoptoclick-style identification has been largely defeated in practice [25], but Pugliese *et al.* later presented counter-arguments from data that such fingerprinting is still an effective threat [44].

Content Blocking. Published countermeasures against user tracking can be broadly categorized as either blocking tracking-related content (e.g., ads) before they enter the browser or changing browser implementations to mitigate unwanted effects from such content. As most ad and tracker blocking currently depends by filter lists, filter list improvements and alternatives are a frequent research topic. Gugelmann *et al.* used large-scale traffic analysis (15k users on a campus network) to train a machine classifier of privacy-invasive tracking services, compared it to popular filter lists, and presented it as a mechanism for updating these lists faster and more effectively than the current crowd-sourced model [26]. The PageGraph instrumentation system has been used to demonstrate the effectiveness and efficiency of ad blocking via machine-learning trained on page graph data [29], to improve filter lists for non-English-speaking populations [47], and to detect filter list evasions in the wild [20]. Hu *et al.* analyzed the interconnectedness (or "tangle factor") of first-party sites embedding the same third-party tracking content using real-world browsing data from volunteers in order to assess ad blocker effectiveness and to drive automatic partitioning of first-party sites into isolated multi-account containers [28].

Browser Policies & Mechanisms. page-length storage belongs to another category of tracking countermeasure research, which focuses on evaluating and enhancing built-in browser security policies. Hypothetical discussions of blocking third-party storage, and of potential collusion by third-parties to work around it, predate the modern era of tracking research [30]. Bauer *et al.* demonstrated practical formal browser security using a taint-analysis and data-flow policy enforcement engine build into Chrome 32; the system could be used to enforce classic browser policies (SOP, CSP) or prototype new ones [19]. Pan *et al.* prototyped an full replacement of the traditional SOP with a hierarchy of nested security principals, each layer able only to increase, not decrease, restrictions on tracking [40]. Fingerprinting countermeasures that involve injecting randomness into known or suspected entropy sources to disrupt stateless tracking include Privaricator [37] and FPRandom [31]. Yu *et al.* described an elegantly generalized approach to tracking prevention at the data flow level using *k*-Anonymity, deployed in the privacy-focused Cliqz browser [51]. Our approach to quantifying tracking potential is loosely inspired by this data flow approach to defining privacy.

8 Conclusion

Our work addresses the lose-lose dilemma presented to browser developers by third-party storage: maintain the status quo and enable mass user tracking, or block third-party storage and break a significant amount of the useful web. Practical experience suggested it was rare for third-party content to actually need persistent storage to provide desirable functionality to the user. We exploited this insight to design page-length storage, and the results show that a win-win (or at least a win-nearly-always-win) solution is possible to the old lose-lose dilemma. We share our contributions with the browser research and development community: the conceptual design of page-length storage, a novel solution to the third-party state management problem in browsers; our metrics for comparing the privacy and compatibility impact of storage policy changes; our working prototype, made available as open source patches to Chromium, along with our crawl dataset.

References

[1] Browser Market Share Worldwide (Sept 2020). https://gs.statcounter.com/browser-market-share#monthly-202009-202009-bar, 2020. Accessed: 2020-10-12.

[2] Chromium Blog: Building a more private web: A path towards making third party cookies obsolete. https://blog.chromium.org/2020/01/building-more-private-web-path-towards.html, 2020. Accessed: 2020-10-12.

[3] Cookie Status. https://www.cookiestatus.com/, 2020. Accessed: 2020-10-12.

[4] Download New Microsoft Edge Browser | Microsoft. https://www.microsoft.com/en-us/edge, 2020. Accessed: 2020-10-12.

[5] EasyList. https://easylist.to/easylist/easylist.txt, 2020. Accessed: 2020-09-17.

[6] Firefox - Protect your life online with privacy-first products — Mozilla. https://www.mozilla.org/en-US/firefox/, 2020. Accessed: 2020-10-12.

[7] GitHub - Brave Browser Implementation. https://github.com/brave/brave-core/blob/master/chromium_src/components/content_settings/core/common/cookie_settings_base.cc#L39, 2020. Accessed: 2020-10-12.

[8] GitHub - disconnectme/disconnect-tracking-protection. https://github.com/disconnectme/disconnect-tracking-protection, 2020. Accessed: 2020-10-12.

[9] Google Chrome - Download the Fast, Secure Browser from Google. https://www.google.com/chrome/, 2020. Accessed: 2020-10-12.

[10] HTML Standard: 11 Web Storage. https://html.spec.whatwg.org/multipage/webstorage.html#the-localstorage-attribute, 2020. Accessed: 2020-10-12.

[11] HTML Standard: 7.5 Origin. https://html.spec.whatwg.org/multipage/origin.html#origin, 2020. Accessed: 2020-10-12.

[12] Intelligent Tracking Prevention | WebKit. https://webkit.org/blog/7675/intelligent-tracking-prevention/, 2020. Accessed: 2020-10-12.

[13] Open-source, prototype implementation of page-length storage and public dataset. https://anonymous.4open.science/r/9cd53044-10d1-4cd0-bbfe-c2611dc657e5/, 2020. Accessed: 2020-10-15.

[14] Public Suffix List. https://publicsuffix.org/list/public_suffix_list.dat, 2020. Accessed: 2020-09-19.

[15] Safari - Apple. https://www.apple.com/safari/, 2020. Accessed: 2020-10-12.

[16] Secure, Fast & Private Web Browser with Adblocker | Brave Browser. https://brave.com/, 2020. Accessed: 2020-10-12.

[17] ACAR, G., EUBANK, C., ENGLEHARDT, S., JUAREZ, M., NARAYANAN, A., AND DIAZ, C. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)* (2014).

[18] ACAR, G., JUAREZ, M., NIKIFORAKIS, N., DIAZ, C., GÜRSES, S., PIESSENS, F., AND PRENEEL, B. FPDetective: Dusting the Web for Fingerprinters. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)* (2013), pp. 1129–1140.

[19] BAUER, L., CAI, S., JIA, L., PASSARO, T., STROUCKEN, M., AND TIAN, Y. Run-time monitoring and formal analysis of information flows in chromium. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)* (2015).

[20] CHEN, Q., SNYDER, P., LIVSHITS, B., AND KAPRAVELOS, A. Detecting Filter List Evasion With Event-Loop-Turn Granularity JavaScript Signatures. In *Proceedings of the IEEE Symposium on Security and Privacy* (May 2021).

[21] ECKERSLEY, P. How Unique Is Your Web Browser? In *International Symposium on Privacy Enhancing Technologies Symposium* (2010).

[22] ENGLEHARDT, S., AND NARAYANAN, A. Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)* (2016).

[23] FALAHRASTEGAR, M., HADDADI, H., UHLIG, S., AND MORTIER, R. Tracking personal identifiers across the web. In *International Conference on Passive and Active Network Measurement* (2016), Springer, pp. 30–41.

[24] FOUAD, I., BIELOVA, N., LEGOUT, A., AND SARAFIJANOVIC-DJUKIC, N. Missed by filter lists: Detecting unknown third-party trackers with invisible pixels. In *PETS 2020-20th Privacy Enhancing Technologies Symposium* (2020).

[25] GÓMEZ-BOIX, A., LAPERDRIX, P., AND BAUDRY, B. Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. In *Proceedings of the International World Wide Web Conference (WWW)* (2018).

[26] GUGELMANN, D., HAPPE, M., AGER, B., AND LENDERS, V. An automated approach for complementing ad blockers' blacklists. *Proceedings on Privacy Enhancing Technologies 2015*, 2 (2015), 282–298.

[27] HALLGREN, K. A. Computing inter-rater reliability for observational data: an overview and tutorial. *Tutorials in quantitative methods for psychology 8*, 1 (2012), 23.

[28] HU, X., AND SASTRY, N. What a Tangled Web We Weave: Understanding the Interconnectedness of the Third Party Cookie Ecosystem. In *Proceedings of the 12th ACM Conference on Web Science* (Southampton, UK, July 2020), ACM.

[29] IQBAL, U., SNYDER, P., ZHU, S., LIVSHITS, B., QIAN, Z., AND SHAFIQ, Z. Adgraph: A graph-based approach to ad and tracker blocking. In *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 65–78.

[30] JACKSON, C., BORTZ, A., BONEH, D., AND MITCHELL, J. C. Protecting Browser State from Web Privacy Attacks. In *Proceedings of the International World Wide Web Conference (WWW)* (2006).

[31] LAPERDRIX, P., BAUDRY, B., AND MISHRA, V. Fprandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In *International Symposium on Engineering Secure Software and Systems* (2017), Springer, pp. 97–114.

[32] LAPERDRIX, P., RUDAMETKIN, W., AND BAUDRY, B. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *Proceedings of the IEEE Symposium on Security and Privacy* (2016).

[33] LI, T.-C., HANG, H., FALOUTSOS, M., AND EFSTATHOPOULOS, P. Trackadvisor: Taking back browsing privacy from third-party trackers. In *International Conference on Passive and Active Network Measurement* (2015), Springer, pp. 277–289.

[34] LIU, C., WHITE, R. W., AND DUMAIS, S. Understanding web browsing behaviors through weibull analysis of dwell time. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval* (2010), pp. 379–386.

[35] MAYER, J. R., AND MITCHELL, J. C. Third-Party Web Tracking: Policy and Technology. In *Proceedings of the IEEE Symposium on Security and Privacy* (2012).

[36] MCHUGH, M. L. Interrater reliability: the kappa statistic. *Biochemia medica: Biochemia medica 22*, 3 (2012), 276–282.

[37] NIKIFORAKIS, N., JOOSEN, W., AND LIVSHITS, B. Privaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the 24th International Conference on World Wide Web* (2015), pp. 820–830.

[38] OLEJNIK, Ł., ACAR, G., CASTELLUCCIA, C., AND DIAZ, C. The Leaking Battery. In *Data Privacy Management, and Security Assurance* (2016), Springer International Publishing.

[39] OLEJNIK, L., MINH-DUNG, T., AND CASTELLUCCIA, C. Selling Off Privacy at Auction. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)* (2014).

[40] PAN, X., CAO, Y., AND CHEN, Y. I Do Not Know What You Visited Last Summer: Protecting Users from Third-party Web Tracking with TrackingFree Browser. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)* (2015).

[41] PAPADOPOULOS, P., KOURTELLIS, N., AND MARKATOS, E. Cookie Synchronization: Everything You Always Wanted to Know But Were Afraid to Ask. In *Proceedings of the International World Wide Web Conference (WWW)* (2019).

[42] PAPADOPOULOS, P., KOURTELLIS, N., AND MARKATOS, E. P. The cost of digital advertisement: Comparing user and advertiser views. In *Proceedings of the 2018 World Wide Web Conference* (2018), pp. 1479–1489.

[43] POCHAT, V. L., VAN GOETHEM, T., TAJALIZADEHKHOOB, S., KORCZYŃSKI, M., AND JOOSEN, W. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)* (2019).

[44] PUGLIESE, G., RIESS, C., GASSMANN, F., AND BENENSON, Z. Long-term observation on browser fingerprinting: Users' trackability and perspective. *Proceedings on Privacy Enhancing Technologies 2020*, 2 (2020), 558–577.

[45] ROESNER, F., KOHNO, T., AND WETHERALL, D. Detecting and Defending Against Third-Party Tracking on the Web. In *Proceedings of the USENIX Symposium on Networked Systems Design & Implementation* (2012).

[46] SCHWENK, J., NIEMIETZ, M., AND MAINKA, C. Same-Origin Policy: Evaluation in Modern Browsers. In *Proceedings of the USENIX Security Symposium* (2017).

[47] SJÖSTEN, A., SNYDER, P., PASTOR, A., PAPADOPOULOS, P., AND LIVSHITS, B. Filter List Generation for Underserved Regions. In *Proceedings of The Web Conference 2020* (2020).

[48] SNYDER, P., TAYLOR, C., AND KANICH, C. Most Websites Don't Need to Vibrate: A Cost-Benefit Approach to Improving Browser Security. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)* (2017).

[49] STOPCZYNSKI, M., TEWS, E., , AND KATZENBEISSER, S. In Depth Evaluation of Redirect Tracking and Link Usage. In *PETS 2020-20th Privacy Enhancing Technologies Symposium* (2020).

[50] YANG, Z., AND YUE, C. A comparative measurement study of web tracking on mobile and desktop environments. *Proceedings on Privacy Enhancing Technologies 2020*, 2 (2020), 24–44.

[51] YU, Z., MACBETH, S., MODI, K., AND PUJOL, J. M. Tracking the trackers. In *Proceedings of the 25th International Conference on World Wide Web* (2016), pp. 121–132.

[52] ZIMMECK, S., LI, J. S., KIM, H., BELLOVIN, S. M., AND JEBARA, T. A Privacy Analysis of Cross-device Tracking. In *Proceedings of the USENIX Security Symposium* (2017), pp. 1391–1408.